Physical Sciences 3

Lectures 11 and 12 – March 12, 2008 – Binary Coding, Digital Circuits, and Magnetism

Reading for Understanding: Chapter 27 s1-6, Chapter 28

DECIMAL → BINARY (transition to)

DECIMAL

base 10 where $10^0 = 1$

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 ← DIGITS

ex: 237 in base 10 form:

$= 200 = 2 \times 100 = 2 \times 10^2$

$+ 30 = 3 \times 10 = 3 \times 10^1$

$+ 7 = 7 \times 1 = 7 \times 10^0$

BINARY

base 2 where $2^0 = 1$

Q: why? only has two options 0 & 1 so less ambiguity than other systems, easier to check errors, can be used to open/close off/on, red/blue – distinguish between two states, also easier to store info. (pixels, pictures)

BINARY EQUIV →

1	1	0	0	0	1	1
64	32	16	8	4	2	1

8 bits = 1 byte

99 − 64 = 35

−32

3

−2

1

−1

0

→ let's look at how to convert the decimal # 99 to binary:

First, write down a short list of 2^n:

2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
256	128	64	32	16	8	4	2	1

Ask, what is the largest power of 2 that can fit into 99?

* see that 64 is the largest 2^n
* so there will be a minimum of 7 bits in the binary equivalent
* for every 2^n that goes into the decimal, give it a 1 and subtract that 2^n from the decimal #

ADDING BINARY

0	0	1	1
+0	+1	+0	+1
00	01	01	10

* 1st bit is carry
* 2nd bit is sum

carry ↑ sum

ex:

		1	1	1 ← carry on top
7		0	1	1
+ 5		0	1	0
12	1	1	0	0

equiv → 2^3 2^2 2^1 2^0

other operations

MULTIPLYING BINARY

just like you do in decimal

5 0000101

×5 0000101 ← do this 1 bit first

101

000 etc... then the second bit

what if we need to check math? →

PARITY CHECK

the last # in the binary is the parity bit, usually indicated by an underline → just tells how many 1s there are: odd # = 1, even # = 0

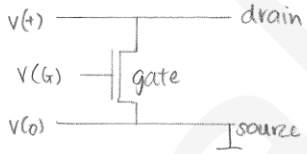
1 1 0 1 0 1 1 0 1 0 ← parity

check rows & columns

TRANSISTORS

nothing but switches! → 0 & 1 off/on open/close

a model:



V(G) + closed

V(G) 0 open

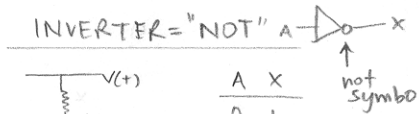
symbolize ▷

** Everything is addition

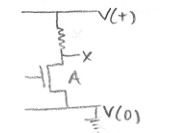
A + B = sum + carry

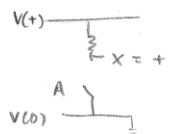
(XOR) (AND)

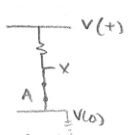
INVERTER = "NOT"



A	X
0	1
1	0

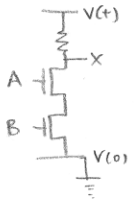






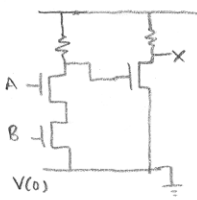
NAND = NOT AND

→ not (A and B)



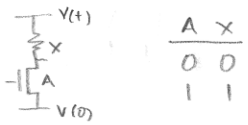
A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

AND = NOT NAND → gives carry



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

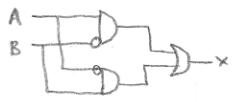
identity



A	X
0	0
1	1

XOR

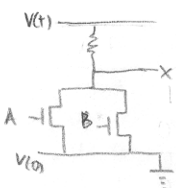
AND OR ○ not



A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

sum bit

NOR ← not (A or B)



A	B	X
0	0	1
0	1	0
1	0	0
1	1	0